# PROGRAMMING DISTRIBUTED APPLICATIONS FOR MOBILE PLATFORMS USING MPI

Iulian VÎRTEJANU[1], Costică NIȚU[2]

*Message Passing Interface (MPI) is library specification meant to facilitate developing distributed applications for high performance computing. MPICH2 is a freely available implementation of the MPI standard. The first part of the paper describes a method for cross-compiling the MPICH2 library for ARM™ platforms (specifically, for Google Android™ OS phones). The second part presents the results of running a distributed application (specifically, π estimation using Monte Carlo statistical method) on two single core ARM™ CPU mobile phones connected via WIFI.*

**Keywords**: message-passing, distributed, ARM™, mobile phones

## 1. Introduction

Even if Android OS is a recent player on the smart phone market, it needed only few years to become a leading platform in the world [1]. Based on Linux kernel, Android does not use the standard *glibc* [2] which accompanies most Linux distributions, but a limited (compared to *glibc*) alternative custom made for mobile phones, alternative named *Bionic*. "The core idea behind Bionic's design is: KEEP IT REALLY SIMPLE." [3]. Bionic fully supports ARM™ architecture.

Due to advance in technology, there will be a lot of powerful mobile devices out there in few years; already multi-core CPU mobile phones (such as Samsung Galaxy S2) are sold.

The paper presents a method to run distributed applications on Android based mobile phones, without relying on *Bionic* C library.

## 2. Estimating $\pi$ using Monte Carlo method

We decided to choose a simple and easily parallelizable algorithm, which is estimating $\pi$ using Monte Carlo integration methods [4].Given a square with sides of $2r$ and a circle with the radius $r$ contained within the square, the area rapport between them will be circle area / square area, thus equal to:

$$\frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

[1] PhD. student., Faculty of Electronics, University POLITEHNICA of Bucharest, Romania, e-mail: viulian@gmail.com

[2] Prof., Faculty of Electronics, University POLITEHNICA of Bucharest, Romania, e-mail: cnitu_upb2001@yahoo.com

If $N$ points are generated randomly within the square, then about $N\pi/4$ should fall within the circle within the square.

If we have an algorithm that generates $N$ points randomly and counts the number of those falling within the circle (let's say using variable$C$), then $\pi$ can be estimated using the formula:

$$\pi \cong \frac{4C}{N}$$

The algorithm is very easy to parallelize since processes do not depend on intermediate results of other processes. If P processes are used, we need one broadcast event to inform the nodes of initial conditions and at the end all processes send back a message to the master containing the results.

**3. MPICH2**

Message Passing Interface (MPI) is library *specification* meant to facilitate developing distributed applications for high performance computing. MPICH2 is a freely available *implementation* of the MPI standard.

MPICH2 contains the library itself (that distributed applications can use to for message-passing between processes) as well as a set of tools that facilitate running processes on different machines, all with a minimal amount of setup.

Two computers were used to test and benchmark the algorithm. They are Windows 7 64bit based machines, running on Intel architecture and connected via Wireless network:

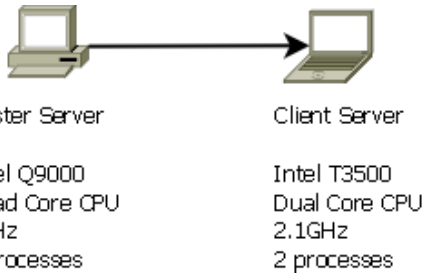


Fig. 1 - Hardware setup for testing purposes

Since the setup involved CPUs having different number of cores, MPICH2 was configured to execute 4 processes on master server and 2 processes on the additional machine. MPICH2 will try to distribute the processes based on the hints indicated in the machine file (a file which describes the IP address of each cluster node and how many processes are to be executed on that specific node).

The most important API methods used are `MPI_Bcast` and `MPI_Reduce` [8] which will send a message to all nodes in the cluster and respectively, will combine the results from all nodes into the master process (rank = 0).

Two variables were used to test the algorithm:

1. Number of processes (from 1 to 12, and then 16 processes)
2. Number of iterations (equal to the total number of points $N$ that the processes had to distribute among themselves and calculate $\pi$ statistically)

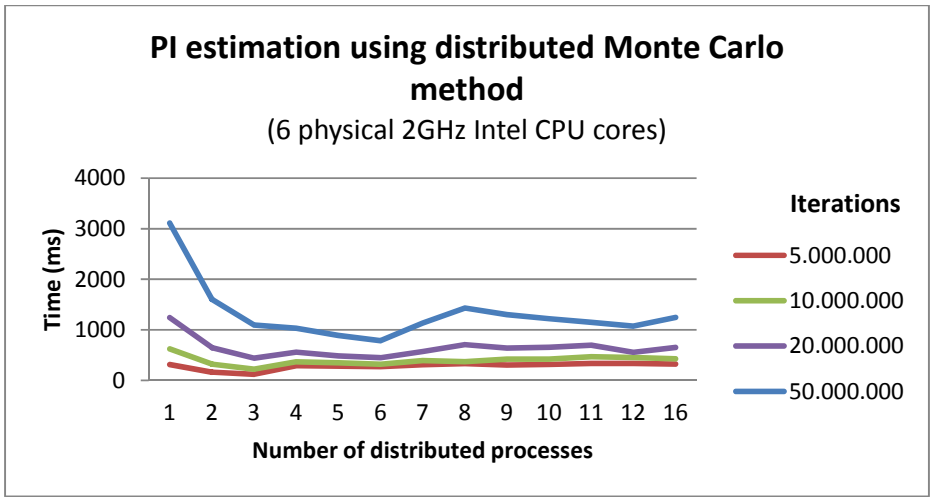


Fig. 2 - Algorithm results on PC based cluster

The results indicate that for a small number of iterations, the CPUs finish the job faster than what it takes the framework to setup the processes and communication channels. To see an impact, we need to increase the number of iterations so that the setup time becomes significantly less than the time framework needs to set up the cluster. Thus, for $N = 20.000.000$ the fastest computation happens when 1 process runs per CPU (six distributed processes). We notice that also at 12 distributed processes (2 processes / CPU) the time needed to estimate $\pi$ is small. This is because for intermediate number of processes, one CPU core will have to handle more than one process which delays the total execution time.

The algorithm works and we could try it out on real mobile phones. Current Android OS current phones run on ARM™ CPUs but are not shipped with a preinstalled native compiler. Tools and distributed application executable code for ARM™ need to be generated on a platform where a C compiler is available. Using a compiler to generate executable code for a different platform different than the one where compiling takes place is a process named *cross-compiling*.

Since MPICH2 is currently distributed for major UNIX operating systems which rely on a full *glibc* canonical library, we decided not to use Android's

Bionic library. The choice was *uClibc* library, an almost complete (compared with *glibc*) library made especially for embedded devices.

**4. Compiling MPICH2 for ARM™ architecture**

To generate executable ARM™ code, an Intel X86 platform (E8500 Intel CPU) running with Ubuntu 11.40 was used.

Requirements:

- *buildroot* [5] – a complete C bundle that allows generating a full Linux distribution for embedded devices. A GNU toolchain was created which allowed cross-compiling to ARM™ executable code on Intel X86 platforms. It contains *uClibc*.
- *MPICH2* [6] – a freely available MPI implementation library that was statically linked to the executable.
- *Unlocked Android phones.* This will void the warranty, but produces already give away tools to unlock the bootloader of the phones allowing root access.

Once the toolchain is generated, MPICH2 can be cross-compiled using command:

```
export PATH=/home/viulian/buildroot-2011.05;%PATH%
export LIBSDIR=/home/viulian/buildroot-2011.05/output/target/usr
CFLAGS="-I/home/viulian/buildroot-2011.05/output/target/usr/include/libxml2 -
DHAVE_DEPRECATED_DNS_FUNCS -march=armv5 -mfpu=vfp -fPIC -fno-exceptions -mlong-
calls -O3 -ffunction-sections -fno-short-enums -fomit-frame-pointer -fno-strict-
aliasing -finline-limit=64 -static" CC=arm-linux-gcc ./configure --prefix=/system
--host=arm-linux --disable-ipv6 --enable-static --with-pm=smpd --with-
device=ch3:sock --disable-f77 --disable-fc
```

The resulting executables files (`mpiexec` and `smpd`) should be copied to the phones within the `/system/xbin/` folder.

Since MPICH2 was not meant to run in Android environment, there are few things that we encountered:

- `$HOME` environment variable has to point to a location not found on the SDCard (since on Android, SDCard is mounted with generic attributes that forbid changing the rights of the files). MPICH2 requires that the configuration files are only readable by the current user.
- `/etc/hosts` files have to indicate the address of each phone in the cluster. Otherwise communication hangs, even if the cluster configuration file specifies the IP of each phone.
- `hostname` command is mandatory to be used to give a name to each phone. Otherwise, the API considers the phone name to be '`localhost`', and communication between the cluster nodes will block indefinitely.

For now, we've only *cross-compiled* the tools that are used to setup the mobile phone cluster. The next step is to cross-compile the distributed application. The C code present in the `mcpi.c` file [7] has to be compiled using the following `Makefile` file:

```
CC=/home/viulian/system/bin/mpicc
TARGET=mcpi
all: $(TARGET)
$(TARGET): $(TARGET).o
	$(CC) -static -o $@ $(TARGET).o $(LIBS) $(LDFLAGS)
$(TARGET).o: $(TARGET).c
	$(CC) $(CFLAGS) -c $(TARGET).c
clean:
	rm -f *.o $(TARGET)
```

The executable file `mcpi` was copied on `/system/xbin/` folders of the phones which by default is present in the `$PATH` environment variable. Thus, the cluster will be able to locate the distributed application executable when executing the job.

## 5. Running the distributed application

The cluster was created by two mobile phones running on a single core ARM™ CPU. One was *Qualcomm MSM8255 Snapdragon* 1GHz CPU and the other was *Qualcomm MSM7227* 600 MHz CPU. The best results are obtained when all nodes in the cluster are identical in performance (otherwise there's visible performance degradation – since other nodes will wait for the slower node one to finish computing). Since the 1GHz CPU is almost twice as fast as the lower end one, we decided to run 2 processes on it.

The results indicate that the fastest computation happens when 7 processes are running. It is so because at the given CPU speeds and distribution (2 processes per faster core, 1 processes for slow one) MPICH2 selected to run 5 processes on the fastest CPU, and 2 on the slow one. This minimized the wait time since in other possible distribution of the processes puts more load on one of the phones and the whole wait time increases.

The results of running the application on the cluster:

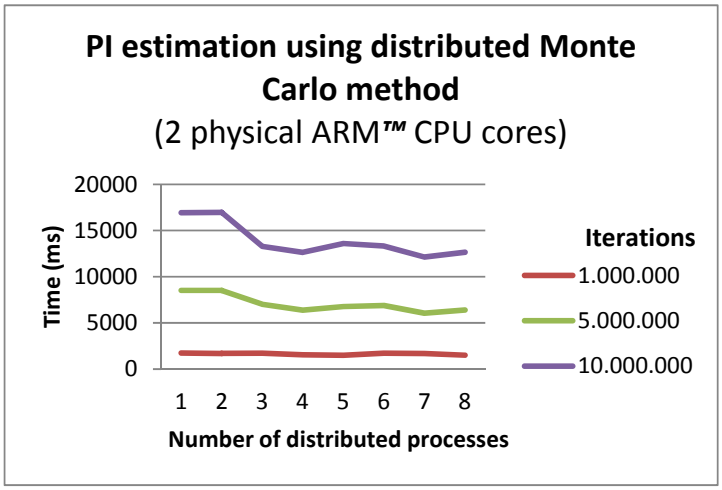


Fig. 3 - Results of running distributed application on mobile phone cluster

## 6. Conclusion

This paper shows that running distributed applications on mobile phones is possible, and they can be used as a platform to foster new and innovative ideas.

Some limitations were observed. There is a need for new standard which allows for dynamic nodes to join the cluster and handle units of works automatically. MPICH2 framework does not allow these; the cluster configuration (machine addresses, processes / machines) has to be defined prior to the job being started. Another major limitation is that for a phone to be part of a dynamic network, additional security measures are needed, such as a VPN. However, a secure VPN is currently not sustainable since it requires additional processing power (battery) and also the overhead added to each TCP/IP packet which slows down the message-passing communications.

With the advent with mobile technology (better batteries, faster network, faster CPUs) having a dynamic cluster of mobile phones will create new business ideas and improve the quality of life. An application idea would be the "family cluster", where family members mobile phones are personal PCs are part of the same cluster with improved privacy and reliability, without the need for a third party service provider acting like master hub.

## R E F E R E N C E S

[1] *Gartner*, Market Share Analysis: Mobile Devices, Worldwide, 1Q11, Retrieved from http://www.gartner.com/it/page.jsp?id=1689814

[2] *A. Ulvesand, D. Eriksson*, Native code on Android, 2011, Retrieved from http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand11/Group1Mads/andreas.ulvesand.daniel.eriksson.report.pdf

[3] Bionic C Library Overview, Retrieved from http://www.netmite.com/android/mydroid/1.5/bionic/libc/docs/OVERVIEW.TXT

[4] *A. Doucet, N. De Freitas, N. Gordon*, Sequential Monte Carlo methods in practice, Birkhäuser, 2001

[5] *Peter Korsgaard*, Buildroot: making Embedded Linux easy, Retrieved from http://buildroot.uclibc.org/

[6] *MPICH2*, Retrieved from http://www.mcs.anl.gov/research/projects/mpich2/

[7] *mcpi.c*, Retrieved from http://hex.ro/tracker/EasyTracker.php?id=47

[8] *Joseph D. Sloan*, High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI, O'Reilly, 2004